A parser for parsing data packets

Related applications

The present rule is a group of five patent applications having the same priority date. Application PCT/SG02/-----relates to an switch having an ingress port which is configurable to act either as eight FE (fast Ethernet) ports or as a GE (gigabit Ethernet port). The present application relates to a parser suitable for use in such as switch. Application PCT/SG02/----- relates to a flow engine suitable for using the output of the parser to make a comparison with rules. Application PCT/SG02/----- relates to monitoring bandwidth consumption using the results of a comparison of rules with packets. Application PCT/SG02/----- relates to a combination of switches arranged as a stack. The respective subjects of the each of the group of applications have applications other than in combination with the technology described in the other four applications, but the disclosure of the other applications of the group is incorporated by reference.

Field of the invention

The present invention relates to a parser system for parsing multiple data packets to extract information from them. In a particular example, the parser system may be employed in a switch such as an Ethernet switch to parse received data packets and thereby obtain information which is required for processing the packet by the queue management system and switching fabric.

Background of Invention

Recent advances in Internet technology have changed the way we exchange information. Ubiquitous use of the Internet has led to the idea of convergence, The different types of data (e.g. video, sounds, pictures and text) must traverse the same network, and this has given rise to a plethora of protocols

which aim to transmit real time and data traffic on a single network with quality of service support.

Chief among these protocols are DiffServ, IntServ and MPLS, which each require packet classification (i.e. determination of the packet's type) in real time when it is received. The first step in this classification is to extract relevant bytes from a packet, "parsing". This is described in Chapter I of "Computer Networks", Andrew S Tanenbaum, Prentice Hall 2nd Ed, 1988.

The parsing operation includes determining whether the packet includes tags. For example, conventionally Ethernet packets may include a VLAN (virtual local area network) tag – i.e. a tag which indicates a VLAN associated with the packet. A VLAN tag is conventionally 4-bytes inserted into the Ethernet frame between the Source MAC Address field and the Length/Type field. The first 2 bytes of the VLAN tag are always set to a value of 0x8100, while the second two bytes are control information (user priority field, canonical format indicator and VLAN identifier).

Another type of tag is defined by the SNAP protocol (subnetwork access protocol), which was introduced to allow older frames and protocols to be encapsulated in a Type 1 LLC header so making any protocol 'pseudo-IEEE compliant'. The SNAP tag (or "snap encapsulation") is placed directly after the standard length/type field of the Ethernet packet (which always takes a value less than or equal to 1500), and has AA-AA as its first two bytes. A packet containing a SNAP tag is called SNAPped.

Generally, an Ethernet packet is made up of levels of nested data, known as layers. Data which is interpreted directly by a machine is called "layer 1", or physical layer, data. "Layer 2", or data link layer, data is LAN (local area network) data, such as MAC (media access control) data uniquely identifying an adapter on the LAN. Within the "layer 2" packet may be "layer 3", or

network layer, data defining among other things the IP source address and destination address of the packet. Within the layer 3 packet may be "layer 4" data, or transport layer data, e.g. TCP (transmission control protocol) data.

In view of the great variety of protocols which may be encountered, it would be useful to provide a parsing technique which is highly flexible.

Additionally, there are a variety of circumstances in which it would be useful to parse a plurality of concurrently received data packets. Such circumstances are not limited to Ethernet applications, but, taking Ethernet applications as an example, in a co-pending pending application referred to above, the present inventors propose a configurable Ethernet switch which can function both as a Fast Ethernet and as a Gigabit Ethernet switch, in order to facilitate the transition from FE to GE Ethernet. A data port can be operated as eight FE MAC interfaces or alternatively as a single GE MAC interface. In the former case, it may happen that the eight FE interfaces receive packets at the same time. It would be possible to provide sufficiently large buffers at the input port that all of these packets are completely received before the processing of any one of the packets begins, but this increases the cost of the buffers required. It would instead be useful to be able to process all of the packets concurrently ("on-the-fly") as they are received to reduce the buffering requirement.

Summary of the Invention

A first aspect of the invention proposes in general terms that a parser system is arranged to receive a data stream having interleaved sections derived from a plurality of different packets, and to extract data from each section as it arrives. The parser system receives information about each of the sections of data defining which packet it relates to, and employs this information to identify data to be extracted from the data stream.

A second aspect of the invention relates to a parser system having a number of programmable registers which store data for the parser, the parser system receiving a data stream and extracting data from it based on offset information stored in the programmable registers.

In either aspect of the invention, the parser system preferably includes a scanning section which identifies the location of major structural features of data in the data stream (e.g. a location where one of the layers of data commences), and at least one parser unit which uses the output of the scanning section and offset information (at least partly from the programmable registers in the case of the second aspect of the invention) to extract the data. The offset information identifies the offset of the data to be extracted from the location of the structural features.

The scanning section uses the received information about the data stream, and also examines the data itself to identify characteristics of packets in the data stream. For example, in addition to determining the location of the start of any one or more of layer 2, layer 3 and/or layer 4 data in the packets, it may further be able to identify if the packet is VLAD tagged and/or SNAPed.

There may be two parser units, one of which extracts data according to predefined offsets and the structural data from the scanning section, and the other extracting data according to the structural data from the scanning section and offsets defined by the programmable registers.

Brief Description of The Figures

Preferred features of the invention will now be described, for the sake of illustration only, with reference to the following figures in which:

Fig. 1 shows schematically the operation of a parser system which is an embodiment of the invention;

Fig. 2 shows schematically how key extraction is performed by the second parser and combiner of the embodiment of Fig. 1;

Fig. 3 shows the structure of 4 types of packets to be parsed by the embodiment; and

Fig. 4 shows the parser system of Fig. 1 as a circuit diagram.

Detailed Description of the embodiments

Referring firstly to Fig. 1, the operation of the embodiment is shown schematically.

The embodiment processes a data steam having packets 1 containing layer 2 data 2 (starting at bit 0) and layer 3 data 3 In fact the packet will also contain layer 4 data, but for the purposes of this embodiment this may be treated simply as part of the level 3 data. The position of the start of the layer 4 data is given by a field which is part of the layer 3 data.

The data stream enters Fig. 1 as a series of sections of predetermined length. Preferably the data stream consists of a series of concurrent packets interleaved. For example, there may be up to 8 packets, which are divided into sections (e.g. of 8 bytes at a time), the sections of different data packets being interleaved. The steps of Fig. 1 are performed on one of these sections at any time, using information about which of the packets the section comes from. This means that there is no need to buffer the entire data stream as it arrives.

The first step (step 5) of the operation is to determine the layer 3 start offset, and whether the data is VLAN tagged or SNAPed. To begin with, for each 8 bytes received the algorithm calculates various variables as follows. Firstly, it updates a count variable (*length*) which indicates the number of bytes of the

packet received so far, by adding the number of new bytes to the previous value of *length*. A variable *index* is defined as the largest integer which is no greater than *length* divided by 8. A variable *offset* is then defined as *length* modulus 8.

Fig. 3 shows the variables *index* and *offset* for 4 types of packet, labelled (a), (b), (c) and (d) having the data shown in byte locations marked by the row marked as "byte number".

- Packet type (a) is not VLAN tagged or snapped, and layer 3 starts at byte 14.
- Packet type (b) is VLAN tagged (so that bytes 12 and 13 are 0x8100 (a hex notation) and layer 3 starts at byte 18.
- Packet type (c) is SNAPped with snap encapsulation starting at byte 14, and layer 3 data starts at byte 22. The bytes at positions 14 to 19 are 0xAA-AA-03-00-00-00.
- Packet type (d) is SNAPped with snap encapsulation starting at byte 18 and also VLAN tagged (so that bytes 12 and 13 are 8100), and layer 3 data starts at byte 26. The bytes at positions 18 to 23 are 0xAA-AA-03-00-00-00.

To determine the position of the L3 start, the following steps are performed at a time when the variable *length* is such that index is 1:

- Check the bytes at offset 4 and 5. If they are not 0x8100 and not less than 1500, then the byte is type (a), and layer 3 starts at byte 14.
- Otherwise, if the bytes at offset 4 and 5 are 0x8100, then the packet is tagged (the packet must be type (b) or type (d)). Set a variable *tagged* to be equal to 1.

- Otherwise, if the bytes at offset 4 and 5 are less than or equal to 1500, and the bytes at offsets 6 and 7 are 0xAA-AA, then the packet must be snapped. Set a variable *snapped* to be equal to 1.
- Otherwise, the packet is in an unknown protocol.

When the next section of the data packet arrives, so that the variable *length* is such that index is 2:

- Check the bytes at offsets 0 and 1. If they are greater than 1500 and *tagged*=1, then the packet is type (b) and layer 3 begins at byte 18.
- Otherwise, if the bytes at offsets 0 and 1 are less than or equal to 1500 and *tagged*=1, then set *snapped*=1.
- If *tagged*=1 and *snapped*=1 and the bytes at offsets 2 to 7 are AA-AA-03-00-00-00, then the packet is type (d), and layer 3 starts at byte 22.
- Otherwise, if *tagged*=0 and *snapped*=1 and the bytes at offsets 0 to 3 are 0x03-00-00-00, then the packet is type (c), and layer 3 starts at byte 22.
- Otherwise, the protocol is unknown.

Referring once more to Fig. 1, once the positions of the start of the layer 3 (and other layers) are known, the section of the data stream is passed to a first parser 7 and to a second parser 9 as discussed below. Note that the step 5 operation, and the first parser 7 and second parser 9 operations are performed on one of these sections at any time. In this case, the step 5 operation uses section identity information identifying which packets the section of data belongs to, and for example in the case that there are multiple packets maintains a set of variables (e.g. variable *length*) for each of those packets. In the processing of a section of the data stream derived from a given packet, step 5 involves updating the variables for the corresponding packet. The parsers 1 and 2 do not have to know this information however.

The first parser 7 extracts data from the packet according to positions defined by a set of registers 8. For example, when 8 bytes are to be extracted, 8 registers (labelled Offset reg 1, ..., Offset reg 8) are used. Each register holds an indication ("L2/L3") of whether data is to be extracted from the layer 2 or laer 3 data, and also an offset indicating which bytes are to be extracted relative to this starting positions of those layers. In this way the first parser 7 is able to extract local keys. The extracted local keys are compared in an AND operation 11 with 8 16-bit masks 15 (each of the 8 registers extracts 16 bits). The same 8 16-bit masks 15 are compared with 8 16-bit rules 17 by an AND operation 13. The results of the AND operations 13 and 11 are compared in step 19 to produce 8 1-bit results.

Meanwhile the second parser 9 receives the same data stream and the results of the determination of the start of the layers, and extracts a set of 8 bits determined by 8 key selection registers 23. The outputs of the second parser 9 are compared with those of the compare operation 19 in a step 21.

The operation of the second parser 9 and of the combine unit 21 is shown in Fig. 2. The upper portion of Fig. 2 shows the conventional structure of a data packet, starting with layer 2 data ("L2 info"), then layer 3 data ("L3 info"), then layer 4 data ("L4 info"). Using the results of the layer position determination algorithm, the second parser 9 is fed selected ones of the bytes as shown of Fig. 2. According to the outputs of programmable selector 23, the MUX multiplex units 25 output one of their inputs. These are fed to further MUX multiplex units 27, 29. The MUX units 27 receive other portions of the data packet, and also output of the first parser 7. The MUX units 29 receive the respective outputs of the MUX units 27, and also of the respective MUX units 25. MUX units 27, 29 are controlled based on selection signals sel[0] 31, sec[3] 31, which also come from the programmable registers 23. The result is the extracted key for the flow engine.

Referring to Fig. 1 again, a combination 21 of the outputs of the compare operation 19 and the key for the flow engine is made, to generate a final key. The uses of this key will be clear to a skilled reader, as will the exact operation of the two parsers.

Fig. 4 shows the layout of a parser system circuit 35 for implementing the steps of Fig. 1 in the context of an Ethernet switch. The parser system circuit 35 operates on 8 bytes at a time, and has an input interface 41 which receives inputs from a buffer rx_ififo 39 which receives packets from the pins of the Ethernet switch, and also from a MAC interface rx_max_ififo 37 which provides control information including an index identifying the packet description associated with the corresponding packet (this constitutes the section identification information discussed above). The step 5 operation of Fig. 1 is implemented by a unit 43, and the results transmitted *inter alia* to a unit 45 which functions as the first and second parser and performs the combinations shown in Fig. 1 to generate the final key. The unit 45 receives the other data it requires, such as the data of registers 9, 17 and 15 of Fig. 2, from a register file 47. The parser puts all the information for each stream of data (i.e. for each of the concurrent packets) within the packet descriptor for the corresponding packet. By operating on 8 bytes at a time with 2 cycles per processing step the parser is able to manage 8 FE streams.

The output of the unit 45 passes to an output interface 49 of the parser parser_mem_iface, which in turn passes it to other components of the Ethernet switch, in particular to a memory manager rx_mem_mgr 51. Note that all the circuitry of the parser system circuit 35 is preferably implemented on a single integrated circuit.